Varunesh Goyal
140070006

Digital Systems Lab – ALU – Report1

The code for **ALU (top-domain)**:

```
library work;
use work.ee214.all;
entity ALU is
      port (a0,a1,a2,a3,a4,a5,a6,a7,
            b0,b1,b2,b3,b4,b5,b6,b7, in0, in1: in bit;
            y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
end entity;
architecture basic of ALU is
      signal
ad0,ad1,ad2,ad3,ad4,ad5,ad6,ad7,add_cin,o1,o2,o0,o3,o4,o5,o6,o7,for_shift,
            lsh0,lsh1,lsh2,lsh3,lsh4,lsh5,lsh6,lsh7,
            rsh0,rsh1,rsh2,rsh3,rsh4,rsh5,rsh6,rsh7,
            sh0,sh1,sh2,sh3,sh4,sh5,sh6,sh7,
            h0,h1,h2,h3,h4,h5,h6,h7,
            s0,s1,s2,s3,s4,s5,s6,s7 : bit;
begin
            --first to decide whether to add or subtract...deciding the
initial cin
            add_cin <= (not in1) and in0;
            --now to change input to adder dependhing on cin...(takes 2s
complement if required)
            xb0 : XOR1 port map (a=>b0, b=>add_cin, c=>ad0);
            xb1 : XOR1 port map (a=>b1, b=>add_cin, c=>ad1);
            xb2 : XOR1 port map (a=>b2, b=>add_cin, c=>ad2);
            xb3 : XOR1 port map (a=>b3, b=>add_cin, c=>ad3);
            xb4 : XOR1 port map (a=>b4, b=>add_cin, c=>ad4);
            xb5 : XOR1 port map (a=>b5, b=>add_cin, c=>ad5);
            xb6 : XOR1 port map (a=>b6, b=>add_cin, c=>ad6);
            xb7 : XOR1 port map (a=>b7, b=>add_cin, c=>ad7);

            add : eightBitAdder port map (a0=>a0, a1=>a1,
                              a2=>a2, a3=>a3,
                              a4=>a4, a5=>a5,
                              a6=>a6, a7=>a7,
                              b0=>ad0, b1=>ad1,
                              b2=>ad2, b3=>ad3,
                              b4=>ad4, b5=>ad5,
                              b6=>ad6, b7=>ad7,
                              cin=>add_cin,
                              y0=>o0, y1=>o1,
                              y2=>o2, y3=>o3,
                              y4=>o4, y5=>o5,
                              y6=>o6, y7=>o7);
            --this completes both addition and subtraction


            --first use the demux
            demux : Demux38 port map(x0=>b0, x1=>b1, x2=>b2,
                              y0=>sh0, y1=>sh1, y2=>sh2, y3=>sh3, y4=>sh4,
y5=>sh5, y6=>sh6, y7=>sh7);
```

```
            for_shift <= b3 or b4 or b5 or b6 or b7;


                  --now for left shift...xis are input, ais are demux
input, yis are output
                  lefts : leftShift8 port map (x0=>a0, x1=>a1, x3=>a3,
x2=>a2,x4=>a4,x5=>a5,x6=>a6, x7=>a7,
                              a0=>sh0, a1=>sh1, a3=>sh3,
a2=>sh2,a4=>sh4,a5=>sh5,a6=>sh6, a7=>sh7,

      y0=>lsh0,y1=>lsh1,y2=>lsh2,y3=>lsh3,y4=>lsh4,y5=>lsh5,y6=>lsh6,y7=>ls
h7);

                  --now for right shift
                  rights : rightShift8 port map (x0=>a0, x1=>a1, x3=>a3,
x2=>a2,x4=>a4,x5=>a5,x6=>a6, x7=>a7,
                              a0=>sh0, a1=>sh1, a3=>sh3,
a2=>sh2,a4=>sh4,a5=>sh5,a6=>sh6, a7=>sh7,

      y0=>rsh0,y1=>rsh1,y2=>rsh2,y3=>rsh3,y4=>rsh4,y5=>rsh5,y6=>rsh6,y7=>rs
h7);

            mux1: mux168 port map (a0=>rsh0, a1=>rsh1, a3=>rsh3,
a2=>rsh2,a4=>rsh4,a5=>rsh5,a6=>rsh6, a7=>rsh7,
                              b0=>lsh0, b1=>lsh1, b3=>lsh3,
b2=>lsh2,b4=>lsh4,b5=>lsh5,b6=>lsh6, b7=>lsh7,

      y0=>h0,y1=>h1,y2=>h2,y3=>h3,y4=>h4,y5=>h5,y6=>h6,y7=>h7,
                              s=>in0);


            --if b>=8, output is zero always...
            s0<=(not for_shift) and h0;
            s1<=(not for_shift) and h1;
            s2<=(not for_shift) and h2;
            s3<=(not for_shift) and h3;
            s4<=(not for_shift) and h4;
            s5<=(not for_shift) and h5;
            s6<=(not for_shift) and h6;
            s7<=(not for_shift) and h7;

      finalmux: mux168 port map (a0=>o0, a1=>o1, a3=>o3,
a2=>o2,a4=>o4,a5=>o5,a6=>o6, a7=>o7,
                              b0=>s0, b1=>s1, b3=>s3,
b2=>s2,b4=>s4,b5=>s5,b6=>s6, b7=>s7,

      y0=>y0,y1=>y1,y2=>y2,y3=>y3,y4=>y4,y5=>y5,y6=>y6,y7=>y7,
                              s=>in1);


end basic;
```

The code for **Testbench :**

```
library std;
use std.textio.all;

entity Testbench is
end entity;
architecture Behave of Testbench is
      signal a0,a1,a2,a3,a4,a5,a6,a7, b0,b1,b2,b3,b4,b5,b6,b7, in0, in1 :
bit := '0';
      signal y : bit_vector (7 downto 0);
      signal error_flag : bit := '0';
      component ALU
            port (a0,a1,a2,a3,a4,a5,a6,a7,
                  b0,b1,b2,b3,b4,b5,b6,b7, in0, in1: in bit;
                  y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
      end component;
      function bitvec_to_str ( x : bit_vector ) return String is
            variable L : line ;
            variable W : String (1 to x ' length ) :=( others => ' ') ;
            begin
                  write (L , x ) ;
                  W ( L . all ' range ) := L . all ;
                  Deallocate ( L ) ;
            return W ;
      end bitvec_to_str ;
begin
      process
      file f : text open read_mode is "outputs.txt" ;
      file g : text open write_mode is "error_log.txt" ;
      variable a : bit_vector (7 downto 0) ;
      variable b : bit_vector (7 downto 0) ;
      variable op_code : bit_vector (1 downto 0) ;
      variable L : line ;
      variable expected_output : bit_vector (7 downto 0);

      begin
            while not endfile(f) loop
                  readline(f,L);
                  read(L,a);
                  read(L,b);
                  read(L,op_code);
                  read(L,expected_output);
                  a7 <= a(7);
                  a6 <= a(6);
                  a5 <= a(5);
                  a4 <= a(4);
                  a3 <= a(3);
                  a2 <= a(2);
                  a1 <= a(1);
                  a0 <= a(0);
                  b7 <= b(7);
                  b6 <= b(6);
                  b5 <= b(5);
                  b4 <= b(4);
                  b3 <= b(3);
                  b2 <= b(2);
```

```
                  b1 <= b(1);
                  b0 <= b(0);
                  in0 <= op_code(0);
                  in1 <= op_code(1);

                  wait for 10 ns ;
            --this is for the computation of the output by the circuit

                  if(not (y = expected_output)) then
                        write(g,bitvec_to_str(a));
                        write(g,bitvec_to_str(b));
                        write(g,bitvec_to_str(op_code));
                        write(g,bitvec_to_str(y));
                        write(g,bitvec_to_str(expected_output));
                        write(g," ");
                        error_flag <= '1';
                  end if;

                  assert (y = expected_output)
                        report "Error"
                        severity error;
            end loop ;

            assert (error_flag = '0') report "Test completed. Errors
present. See error_log.txt"
                  severity error;
            assert (error_flag = '1') report "Test completed.Successful!!!"
                  severity note;
            wait ;
      end process ;
      dut : ALU port map (a7 => a7,
                  a6 => a6,
                  a5 => a5,
                  a4 => a4,
                  a3 => a3,
                  a2 => a2,
                  a1 => a1,
                  a0 => a0,
                  b7 => b7,
                  b6 => b6,
                  b5 => b5,
                  b4 => b4,
                  b3 => b3,
                  b2 => b2,
                  b1 => b1,
                  b0 => b0,
                  in0 => in0,
                  in1 => in1,
                  y7 => y(7),
                  y6 => y(6),
                  y5 => y(5),
                  y4 => y(4),
                  y3 => y(3),
                  y2 => y(2),
                  y1 => y(1),
                  y0 => y(0));
end Behave ;
```

The codes for the package ee214, and other entities can be found in the appendix at the end.

The python script is as follows:

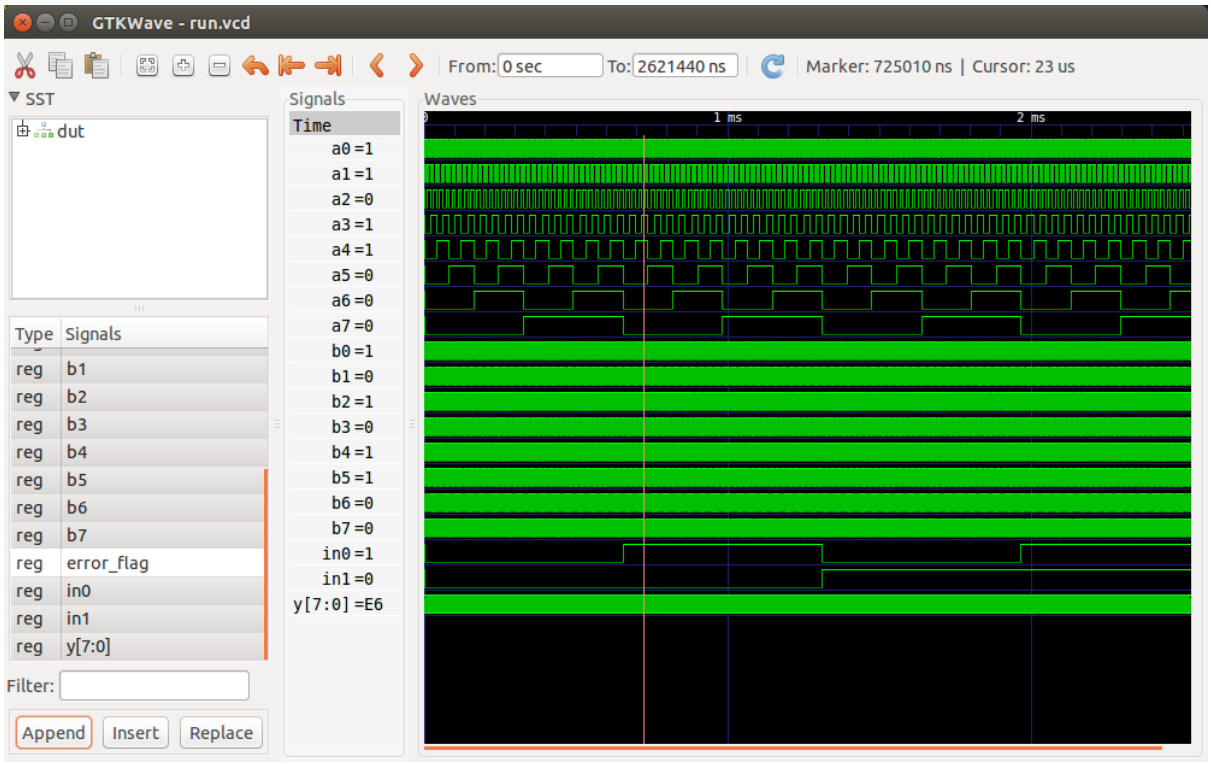
```
import string
f = open("outputs.txt",'w')
string=""
for i in range(256):
      for j in range(256):
            string=""
            string+='{:08b}'.format(i) + " " + '{:08b}'.format(j) + " "
+"00"+ " " + '{:08b}'.format((i+j)%256) +"\n"
            f.write(string)

for i in range(256):
      for j in range(256):
            string=""
            string+='{:08b}'.format(i) + " " + '{:08b}'.format(j) + " "
+"01"+ " " + '{:08b}'.format((i-j)%256) +"\n"
            f.write(string)

for i in range(256):
      for j in range(256):
            string=""
            string+='{:08b}'.format(i) + " " + '{:08b}'.format(j) + " "
+"10"+ " " + '{:08b}'.format((i>>j)%256) +"\n"
            f.write(string)

for i in range(256):
      for j in range(256):
            string=""
            string+='{:08b}'.format(i) + " " + '{:08b}'.format(j) + " "
+"11"+ " " + '{:08b}'.format((i<<j)%256) +"\n"
            f.write(string)
```

The waveform as seen in GTKWAVE :



The terminal window showing all compilations and running of testcases successfully done:

```
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a ee214.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a XOR1.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a bitAdder.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a eightBitAdder.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a Demux-3-8.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a left_shift8.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a right_shift8.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a Mux-2-1.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a Mux-16-8.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a ALU.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -a Testbench.vhd
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -m Testbench
elaborate testbench
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ ghdl -r Testbench --stop-time=10000us --vcd=run.vcd
Testbench.vhd:85:16:@2621440ns:(assertion note): Test completed. Successful!!!
varunesh@varunesh-HP-Pavilion-15-Notebook-PC:~/course/Sem4/EE214$ gtkwave run.vcd

GTKWave Analyzer v3.3.64 (w)1999-2014 BSI

[0] start time.
[2621440000000] end time.
```

Appendix

EE214:

```
library std;
use std.standard.all;

package ee214 is
	component bitAdder is
		port (x, y, cin : in bit;
			s, cout : out bit);
	end component bitAdder;
	component eightBitAdder is
		port (a0,a1,a2,a3,a4,a5,a6,a7,
			b0,b1,b2,b3,b4,b5,b6,b7, cin: in bit;
			y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
	end component eightBitAdder;
	component XOR1 is
		port (a, b: in bit;
			c : out bit);
	end component XOR1;
	component Demux38 is
	port(x0, x1,x2 : in bit;
		y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
	end component Demux38;
	component rightShift8 is
		port (x7,x6,x5,x4,x3,x2,x1,x0,a0,a1,a2,a3,a4,a5,a6,a7: in bit;
			y7,y6,y5,y4,y3,y2,y1,y0 : out bit);
	end component;
	component leftShift8 is
		port (x0, x1,x2,x3,x4,x5,x6,x7,a0,a1,a2,a3,a4,a5,a6,a7: in bit;
			y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
	end component;
	component mux21 is
		port(a,b,s : in bit;
		o : out bit);
	end component mux21;
	component mux168 is
		port (a0,a1,a2,a3,a4,a5,a6,a7,
		b0,b1,b2,b3,b4,b5,b6,b7, s: in bit;
		y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
	end component;
end ee214;
```

XOR1:

```
entity XOR1 is
	port(a,b : in bit;
		c: out bit);
end entity;
architecture one of XOR1 is
begin
	c<= ((not a) and b) or ((not b) and a);
end one;
```

bitAdder:

```
entity bitAdder is
      port(x, y, cin: in bit;
            cout, s: out bit);
end entity;
architecture simple of bitAdder is
begin
      s<= ((not x) and (not y) and cin) or
            ((not x) and y and (not cin)) or
            (x and (not y) and (not cin)) or
            (x and y and cin);
      cout<= (x and y) or (cin and y) or (cin and x);
end simple;
```

eightBitAdder :

```
library work;
use work.ee214.all;
entity eightBitAdder is
      port (a0,a1,a2,a3,a4,a5,a6,a7,
            b0,b1,b2,b3,b4,b5,b6,b7, cin: in bit;
            y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
end entity;
architecture basic of eightBitAdder is
      signal c1,c2,c3,c4,c5,c6,c7,random : bit;
begin


      add0 : bitAdder port map ( x=> a0, y=> b0, cin=> cin, s=> y0,
cout=>c1);
      add1 : bitAdder port map ( x=> a1, y=> b1, cin=> c1, s=> y1,
cout=>c2);
      add2 : bitAdder port map ( x=> a2, y=> b2, cin=> c2, s=> y2,
cout=>c3);
      add3 : bitAdder port map ( x=> a3, y=> b3, cin=> c3, s=> y3,
cout=>c4);
      add4 : bitAdder port map ( x=> a4, y=> b4, cin=> c4, s=> y4,
cout=>c5);
      add5 : bitAdder port map ( x=> a5, y=> b5, cin=> c5, s=> y5,
cout=>c6);
      add6 : bitAdder port map ( x=> a6, y=> b6, cin=> c6, s=> y6,
cout=>c7);
      add7 : bitAdder port map ( x=> a7, y=> b7, cin=> c7, s=> y7,
cout=>random);

end basic;
```

Demux-3-8 :

```
entity Demux38 is
      port(x0, x1,x2 : in bit;
            y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
end entity;
architecture one of Demux38 is
begin
      y0 <= (not x0)    and (not x1)      and (not x2);
      y1 <= (x0)  and (not x1)      and (not x2);
      y2 <= (not x0)    and (x1)    and (not x2);
      y3 <= (x0)  and (x1)    and (not x2);
      y4 <= (not x0)    and (not x1)      and (x2);
      y5 <= (x0)  and (not x1)      and (x2);
      y6 <= (not x0)    and (x1)    and (x2);
      y7 <= (x0)  and (x1)    and (x2);
end one;
```

leftShift8 :

```
entity leftShift8 is
      port (x0, x1,x2,x3,x4,x5,x6,x7,a0,a1,a2,a3,a4,a5,a6,a7: in bit;
            y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
end entity;
architecture one of leftShift8 is
begin
      y0<= x0 and a0;
      y1<= (x1 and a0) or (x0 and a1);
      y2<= (x2 and a0) or (x1 and a1) or (x0 and a2);
      y3<= (x3 and a0) or (x2 and a1) or (x1 and a2) or (x0 and a3);
      y4<= (x4 and a0) or (x3 and a1) or (x2 and a2) or (x1 and a3) or (x0
and a4);
      y5<= (x5 and a0) or (x4 and a1) or (x3 and a2) or (x2 and a3) or (x1
and a4) or (x0 and a5);
      y6<= (x6 and a0) or (x5 and a1) or (x4 and a2) or (x3 and a3) or (x2
and a4) or (x1 and a5) or (x0 and a6);
      y7<= (x7 and a0) or (x6 and a1) or (x5 and a2) or (x4 and a3) or (x3
and a4) or (x2 and a5) or (x1 and a6) or (x0 and a7);
end one;
```

rightShift8 :

```
entity rightShift8 is
      port (x7,x6,x5,x4,x3,x2,x1,x0,a0,a1,a2,a3,a4,a5,a6,a7: in bit;
            y7,y6,y5,y4,y3,y2,y1,y0 : out bit);
end entity;
architecture one of rightShift8 is
begin
      y7<= x7 and a0;
      y6<= (x6 and a0) or (x7 and a1);
      y5<= (x5 and a0) or (x6 and a1) or (x7 and a2);
      y4<= (x4 and a0) or (x5 and a1) or (x6 and a2) or (x7 and a3);
      y3<= (x3 and a0) or (x4 and a1) or (x5 and a2) or (x6 and a3) or (x7
and a4);
      y2<= (x2 and a0) or (x3 and a1) or (x4 and a2) or (x5 and a3) or (x6
and a4) or (x7 and a5);
```

```
      y1<= (x1 and a0) or (x2 and a1) or (x3 and a2) or (x4 and a3) or (x5
and a4) or (x6 and a5) or (x7 and a6);
      y0<= (x0 and a0) or (x1 and a1) or (x2 and a2) or (x3 and a3) or (x4
and a4) or (x5 and a5) or (x6 and a6) or (x7 and a7);
end one;
```

mux-2-1:

```
entity mux21 is
      port(a,b,s : in bit;
            o : out bit);
end entity;
architecture one of mux21 is
begin
      o<=(a and (not s)) or (b and s);
end one;
```

mux-16-8:

```
library work;
use work.ee214.all;
entity mux168 is
      port (a0,a1,a2,a3,a4,a5,a6,a7,
            b0,b1,b2,b3,b4,b5,b6,b7, s: in bit;
            y0,y1,y2,y3,y4,y5,y6,y7 : out bit);
end entity;
architecture one of mux168 is
begin
      bit0 : mux21 port map ( a=> a0, b=> b0, s=>s, o=>y0);
      bit1 : mux21 port map ( a=> a1, b=> b1, s=>s, o=>y1);
      bit2 : mux21 port map ( a=> a2, b=> b2, s=>s, o=>y2);
      bit3 : mux21 port map ( a=> a3, b=> b3, s=>s, o=>y3);
      bit4 : mux21 port map ( a=> a4, b=> b4, s=>s, o=>y4);
      bit5 : mux21 port map ( a=> a5, b=> b5, s=>s, o=>y5);
      bit6 : mux21 port map ( a=> a6, b=> b6, s=>s, o=>y6);
      bit7 : mux21 port map ( a=> a7, b=> b7, s=>s, o=>y7);
end one;
```